File Edit View History Bookmarks Tools Help

Manipulating data in PL/SQL

www.pgrocer.net/Cis50/plsqldata.html

Manipulating data within PL/SQL

Within PL/SQL you can manipulate individual records as shown in the examples below. This is the description of the table that I will be using. I also have created a sequence that I will be using.

SQL CODE:

```
SQL> DESC testplsql;
 Name                            Null?    Type
 ------------------------------- -------- ----
 IDNO                                     NUMBER(3)
 NAME                                     VARCHAR2(20)
 AMT1                                     NUMBER(6,2)
 A_CODE                                   CHAR(2)
 AMT2                                     NUMBER(4)

SQL> CREATE SEQUENCE
  2  testplsql_seq
  3  INCREMENT BY 1
  4  START WITH 111
  5  MAXVALUE 999;

Sequence created.
```

Next, I wrote the code as test_insert to put a record into the file using the sequence number generator for the idno. First, I got the number and stored in as a variable. Then I did the insert. Please remember the / at the end of the code. I get into trouble when I don't. I ran test_insert once, got the results, went in and changed the data and ran it again! Cumbersome!

SQL CODE:

```
SQL> edit test_insert
```

PL/SQL CODE:

```
DECLARE
   v_idno testplsql.idno%TYPE;
BEGIN
   SELECT testplsql_seq.NEXTVAL
   INTO v_idno
   FROM dual;
   INSERT INTO testplsql
   VALUES(v_idno, 'John Smith', 112.12, 'MA', 234);
END;
/
```

SQL CODE:

```
SQL> @ test_insert
Input truncated to 1 characters

PL/SQL procedure successfully completed.

SQL> SELECT * FROM testplsql;
```

Type here to search

9:40 AM
10/29/2019

File Edit View History Bookmarks Tools Help

Manipulating data in PL/SQL

www.pgrocer.net/Cis50/plsqldata.html

```
 A_CODE                                   CHAR(2)
 AMT2                                     NUMBER(4)

SQL> CREATE SEQUENCE
  2  testplsql_seq
  3  INCREMENT BY 1
  4  START WITH 111
  5  MAXVALUE 999;

Sequence created.
```

Next, I wrote the code as test_insert to put a record into the file using the sequence number generator for the idno. First, I got the number and stored in as a variable. Then I did the insert. Please remember the / at the end of the code. I get into trouble when I don't. I ran test_insert once, got the results, went in and changed the data and ran it again! Cumbersome!

SQL CODE:

```
SQL> edit test_insert
```

PL/SQL CODE:

```
DECLARE
   v_idno testplsql.idno%TYPE;
BEGIN
   SELECT testplsql_seq.NEXTVAL
   INTO v_idno
   FROM dual;
   INSERT INTO testplsql
   VALUES(v_idno, 'John Smith', 112.12, 'MA', 234);
END;
/
```

Using the sequence to provide the v_idno.

SQL CODE:

```
SQL> @ test_insert
Input truncated to 1 characters

PL/SQL procedure successfully completed.

SQL> SELECT * FROM testplsql;

     IDNO NAME                      AMT1 A_      AMT2
--------- -------------------- --------- -- ---------
      111 John Smith              112.12 MA       234

SQL> edit test_insert
```

PL/SQL CODE:

```
DECLARE
   v_idno testplsql.idno%TYPE;
BEGIN
   SELECT testplsql_seq.NEXTVAL
   INTO v_idno
   FROM dual;
   INSERT INTO testplsql
   VALUES(v_idno, 'Ann Wilson', 27.28, 'RI', 987);
```

Type here to search

9:42 AM
10/29/2019

Title: Oct 29-9:42 AM (2 of 18)

File Edit View History Bookmarks Tools Help

Manipulating data in PL/SQL

www.pgrocer.net/Cis50/plsqldata.html

```
   INTO v_idno
   FROM dual;
   INSERT INTO testplsql
   VALUES(v_idno, 'Ann Wilson', 27.28, 'RI', 987);
END;
/
```

SQL CODE:

```
SQL> @ test_insert
Input truncated to 1 characters

PL/SQL procedure successfully completed.

SQL> SELECT * FROM testplsql;

     IDNO NAME                      AMT1 A_      AMT2
--------- -------------------- --------- -- ---------
      111 John Smith              112.12 MA       234
      112 Ann Wilson               27.28 RI       987
```

Now, I want to adapt the code so that the user can enter the data rather than having to go into the code and make the changes. To do this I am using ACCEPT statements outside the PL/SQL block to take in the data. Then I am assigning the variable that came in as the initial value to the variables I declared in the DECLARE section. Notice that I do this with the assignment sign (:=) followed by the &name in accept. If it is non numeric, I enclose in quotes. The user is then prompted to enter the data, the data is stored in variables which are used in the insert. The identification number is still being generated by the sequence generator used in the examples above. I ran the PL/SQL twice to input two records.

SQL CODE:

```
SQL> edit test_insert1
```

PL/SQL CODE:

Using the prompt to take in the information to insert in the table.

```
SET VERIFY OFF
ACCEPT in_name PROMPT 'Enter name: ';
ACCEPT in_amt1 PROMPT 'Enter amount one: ';
ACCEPT in_a_code PROMPT 'Enter the state code: ';
ACCEPT in_amt2 PROMPT 'Enter amount two: ';
DECLARE
   v_idno testplsql.idno%TYPE;
   v_name testplsql.name%TYPE :='&in_name';
   v_amt1 testplsql.amt1%TYPE :=&in_amt1;
   v_a_code testplsql.a_code%TYPE :='&in_a_code';
   v_amt2 testplsql.amt2%TYPE :=&in_amt2;
BEGIN
   SELECT testplsql_seq.NEXTVAL
   INTO v_idno
   FROM dual;
   INSERT INTO testplsql
   VALUES(v_idno, v_name, v_amt1, v_a_code, v_amt2);
END;
/
SET VERIFY ON
```

SQL CODE:

```
SQL> @ test_insert1
```

Type here to search

9:44 AM
10/29/2019

File Edit View History Bookmarks Tools Help

Manipulating data in PL/SQL

www.pgrocer.net/Cis50/plsqldata.html

Now, I want to adapt the code so that the user can enter the data rather than having to go into the code and make the changes. To do this I am using ACCEPT statements outside the PL/SQL block to take in the data. Then I am assigning the variable that came in as the initial value to the variables I declared in the DECLARE section. Notice that I do this with the assignment sign (:=) followed by the &name in accept. If it is non numeric, I enclose in quotes. The user is then prompted to enter the data, the data is stored in variables which are used in the insert. The identification number is still being generated by the sequence generator used in the examples above. I ran the PL/SQL twice to input two records.

SQL CODE:

```
SQL> edit test_insert1
```

PL/SQL CODE:

```
SET VERIFY OFF
ACCEPT in_name PROMPT 'Enter name: ';
ACCEPT in_amt1 PROMPT 'Enter amount one: ';
ACCEPT in_a_code PROMPT 'Enter the state code: ';
ACCEPT in_amt2 PROMPT 'Enter amount two: ';
DECLARE
   v_idno testplsql.idno%TYPE;
   v_name testplsql.name%TYPE :='&in_name';
   v_amt1 testplsql.amt1%TYPE :=&in_amt1;
   v_a_code testplsql.a_code%TYPE :='&in_a_code';
   v_amt2 testplsql.amt2%TYPE :=&in_amt2;
BEGIN
   SELECT testplsql_seq.NEXTVAL
   INTO v_idno
   FROM dual;
   INSERT INTO testplsql
   VALUES(v_idno, v_name, v_amt1, v_a_code, v_amt2);
END;
/
SET VERIFY ON
```

SQL CODE:

```
SQL> @ test_insert1
Enter name: Susan French
Enter amount one: 200.75
Enter the state code: MA
Enter amount two: 345

PL/SQL procedure successfully completed.

SQL> SELECT * FROM testplsql;

     IDNO NAME                      AMT1 A_      AMT2
--------- -------------------- --------- -- ---------
      111 John Smith              112.12 MA       234
      112 Ann Wilson               27.28 RI       987
      113 Susan French            200.75 MA       345

SQL> @ test_insert1
Enter name: James Rogers
Enter amount one: 500.12
Enter the state code: RI
Enter amount two: 123

PL/SQL procedure successfully completed.
```

Type here to search

9:45 AM
10/29/2019

Title: Oct 29-9:45 AM (4 of 18)

File Edit View History Bookmarks Tools Help

Manipulating data in PL/SQL

www.pgrocer.net/Cis50/plsqldata.html

```
SQL> SELECT * FROM testplsql;

     IDNO NAME                      AMT1 A_      AMT2
--------- -------------------- --------- -- ---------
      111 John Smith              112.12 MA       234
      112 Ann Wilson               27.28 RI       987
      113 Susan French            200.75 MA       345
      114 James Rogers            500.12 RI       123
```

Now I am going to update the field AMT1 for whatever identification number the user enters. The code first does the ACCEPT to take in the identification number which is assigned as an initial value to v_idno-in. Then I did a select to bring in the idno and amt1 where the record matched the input id stored in v_idno_in. The data that was brought in was stored in v_idno and v_amt1. Next I multiplied v_amt1 by 1.1 and stored the answer in v_new_amt1 Next I did the update to set the amt1 on the file equal to v_new_amt1 where the idno on the file matched the idno that was keyed in and stored in v_idno-in. The update was successful as shown in the output below.

SQL CODE:

```
SQL> edit test_update1
```

PL/SQL CODE:

```
SET VERIFY OFF
ACCEPT in_idno PROMPT 'Enter the idno of the record to change amt1 in: ';
DECLARE
   v_idno_in testplsql.idno%TYPE :=&in_idno;
   v_idno testplsql.idno%TYPE;
   v_amt1 testplsql.amt1%TYPE;
   v_new_amt1 testplsql.amt1%TYPE;
BEGIN
   SELECT idno, amt1
     INTO v_idno, v_amt1
     FROM testplsql
     WHERE idno = v_idno_in;
   v_new_amt1 := v_amt1 * 1.1;
   UPDATE testplsql
     SET amt1 = v_new_amt1
     WHERE idno = v_idno_in;
END;
/
SET VERIFY ON
```

Doing a calculation and an update of the specified record.

SQL CODE:

```
SQL> @ test_update1
Enter the idno of the record to change amt1 in: 113

PL/SQL procedure successfully completed.

SQL> SELECT * FROM testplsql;

     IDNO NAME                      AMT1 A_      AMT2
--------- -------------------- --------- -- ---------
      111 John Smith              112.12 MA       234
      112 Ann Wilson               27.28 RI       987
      113 Susan French            220.83 MA       345
      114 James Rogers            500.12 RI       123
```

Type here to search

9:46 AM 10/29/2019

Title: Oct 29-9:46 AM (5 of 18)

File Edit View History Bookmarks Tools Help

Manipulating data in PL/SQL

www.pgrocer.net/Cis50/plsqldata.html

```
END;
/
SET VERIFY ON
```

SQL CODE:

```
SQL> @ test_update1
Enter the idno of the record to change amt1 in: 113

PL/SQL procedure successfully completed.

SQL> SELECT * FROM testplsql;

     IDNO NAME                      AMT1 A_      AMT2
--------- -------------------- --------- -- ---------
      111 John Smith              112.12 MA       234
      112 Ann Wilson               27.28 RI       987
      113 Susan French            220.83 MA       345
      114 James Rogers            500.12 RI       123
```

The next thing is to delete a record. I am actually going to delete the record and then at the SQL> prompt, I will recall the record. To do this I am going to commit within the code before I do the delete so the rollback will only go back through the delete and restore the record that was deleted. Note that the rollback is issued at the SQL prompt, or I would not be able to see the results of the delete.

SQL CODE:

```
SQL> edit delete1
```

PL/SQL CODE:

```
SET VERIFY OFF
ACCEPT in_idno PROMPT 'Enter the idno of the record to delete: ';
DECLARE
   v_idno_in testplsql.idno%TYPE :=&in_idno;
   v_idno testplsql.idno%TYPE;

BEGIN
   COMMIT;
   DELETE FROM testplsql
     WHERE idno = v_idno_in;
END;
/
SET VERIFY ON
```

SQL CODE:

```
SQL> @ test_delete1
Enter the idno of the record to delete: 113

PL/SQL procedure successfully completed.

SQL> SELECT * FROM testplsql;

     IDNO NAME                      AMT1 A_      AMT2
--------- -------------------- --------- -- ---------
      111 John Smith              112.12 MA       234
```

Type here to search

9:47 AM
10/29/2019

Title: Oct 29-9:47 AM (6 of 18)

File Edit View History Bookmarks Tools Help

Manipulating data in PL/SQL

www.pgrocer.net/Cis50/plsqldata.html

PL/SQL CODE:

```
SET VERIFY OFF
ACCEPT in_id PROMPT 'Enter id #: '
ACCEPT in_name PROMPT 'Enter name: '
ACCEPT in_hrs PROMPT 'Enter hours worked: '
ACCEPT in_payhr PROMPT 'Enter pay per hour: '
DECLARE
  v_pay NUMBER(6,2);
  v_hrs NUMBER(2) :=&in_hrs;
  v_payhr NUMBER(5,2) := &in_payhr;
BEGIN
  v_pay := v_hrs * v_payhr;
  INSERT INTO testpay
  VALUES(&in_id,'&in_name', v_pay);
END;
/
SET VERIFY ON
```

After saving pay2 with the code I wrote, I exited the editor. Then, at the SQL prompt, I executed pay2. The accepts displayed the prompts and I entered in the data. The code then Inserts the data into the table testpay. I selected the contents of testpay to show the insertion of the first record. I then ran pay2 again and a second record was inserted in the table. Notice also the haunting Input truncated...

SQL CODE:

```
SQL> @ pay2
Enter id #: 123
Enter name: John Doe
Enter hours worked: 40
Enter pay per hour: 12

PL/SQL procedure successfully completed.

Input truncated to 13 characters
SQL> SELECT * FROM testpay;

     IDNO NAME                   GROSSPAY
--------- -------------------- ----------
      123 John Doe                    480

SQL> @ pay2
Enter id #: 234
Enter name: Ann Smith
Enter hours worked: 40
Enter pay per hour: 10.75

PL/SQL procedure successfully completed.

Input truncated to 13 characters
SQL> SELECT * FROM testpay;

     IDNO NAME                   GROSSPAY
--------- -------------------- ----------
      123 John Doe                    480
      234 Ann Smith                   430
```

Type here to search

9:48 AM 10/29/2019

Title: Oct 29-9:48 AM (7 of 18)

File Edit View History Bookmarks Tools Help

Introduction to cursors

www.pgrocer.net/Cis50/cursors.html

Cursors in PL/SQL

Oracle supports two types of cursors that can be used in PL/SQL: the implicit cursor and the explicit cursor. The implicit cursor is created and managed by PL/SQL when the executable part of the PL/SQL block processes a SQL statement while the explicit cursor is created and managed by the programmer.

Before I demonstate the use of the cursor, I wanted a table that contained parts of the donor table. I created the donor_part table to hold the name, yrgoal and state. Actually I really wanted the table empty, so I deleted the data as shown below.

SQL CODE:

```
SQL> CREATE TABLE donor_part
  2  (name, yrgoal, state)
  3  AS SELECT name, yrgoal, state FROM donor;

Table created.

SQL> SELECT * FROM donor_part;

NAME                YRGOAL ST
--------------- --------- --
Stephen Daniels       500 MA
Jennifer Ames         400 RI
Carl Hersey               RI
Susan Ash             100 MA
Nancy Taylor           50 MA
Robert Brooks          50 MA

6 rows selected.

SQL> delete from donor_part;

6 rows deleted.

SQL> select * from donor_part;

no rows selected
```

Creating and emptying the table I am going to use.

When using explicit cursor in PL/SQL there are four things that must be accomplished by the programmer.

Please be sure to go over these.

- First, the cursor must be declared.
- Second, the cursor needs to be opened.
- Third, the fetch is used to fetch the results of the query into the variables declared in PL/SQL.
- Fourth, the cursor needs to be closed.

In this PL/SQL block I declared three variables v_name, v_yrgoal and v_state to correspond to the three fields in the new table that I created above. Then I declared a cursor with a select statement which is actually the query to be processed by the cursor.

In the begin, I open the declared cursor and then fetch the first record into the variables that I declared. This will put name, yrgoal, and state from the first record into v_name, v_yrgoal and v_state. Then I put in a WHILE LOOP that will be executed as long as the donor_cursor finds a record. Inside the loop I insert the data into the record into the new table and then fetch the next record. Notice the construction which puts the initializing fetch before the loop. Inside the loop the processing (putting the record into the new file) is done and then the next fetch is executed. The WHILE LOOP is done as long as the fetch is successful and a record is fetched. This is tested with the donor_cursor%FOUND clause. When the loop is complete, the cursor is closed. Looking at the results you can see that there are now six records in the new table that correspond to the six records in the donor table.

Type here to search

9:51 AM
10/29/2019

File Edit View History Bookmarks Tools Help

Introduction to cursors

www.pgrocer.net/Cis50/cursors.html

```
6 rows deleted.

SQL> select * from donor_part;

no rows selected
```

When using explicit cursor in PL/SQL there are four things that must be accomplished by the programmer.

- First, the cursor must be declared.
- Second, the cursor needs to be opened.
- Third, the fetch is used to fetch the results of the query into the variables declared in PL/SQL.
- Fourth, the cursor needs to be closed.

In this PL/SQL block I declared three variables v_name, v_yrgoal and v_state to correspond to the three fields in the new table that I created above. Then I declared a cursor with a select statement which is actually the query to be processed by the cursor.

In the begin, I open the declared cursor and then fetch the first record into the variables that I declared. This will put name, yrgoal, and state from the first record into v_name, v_yrgoal and v_state. Then I put in a WHILE LOOP that will be executed as long as the donor_cursor finds a record. Inside the loop I insert the data into the record into the new table and then fetch the next record. Notice the construction which puts the initializing fetch before the loop. Inside the loop the processing (putting the record into the new file) is done and then the next fetch is executed. The WHILE LOOP is done as long as the fetch is successful and a record is fetched. This is tested with the donor_cursor%FOUND clause. When the loop is complete, the cursor is closed. Looking at the results you can see that there are now six records in the new table that correspond to the six records in the donor table.

SQL CODE:

```
SQL> edit cursor1
```

PL/SQL CODE:

```
DECLARE
  v_name     donor.name%TYPE;
  v_yrgoal   donor.yrgoal%TYPE;
  v_state    donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;   — 1st record
WHILE donor_cursor%FOUND LOOP
  INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;   — all other records
END LOOP;
CLOSE donor_cursor;
END;
/
```

SQL CODE:

```
SQL> @ cursor1
Input truncated to 1 characters

PL/SQL procedure successfully completed.

SQL> select * from donor_part;
```

Type here to search

9:56 AM
10/29/2019

Introduction to cursors

www.pgrocer.net/Cis50/cursors.html

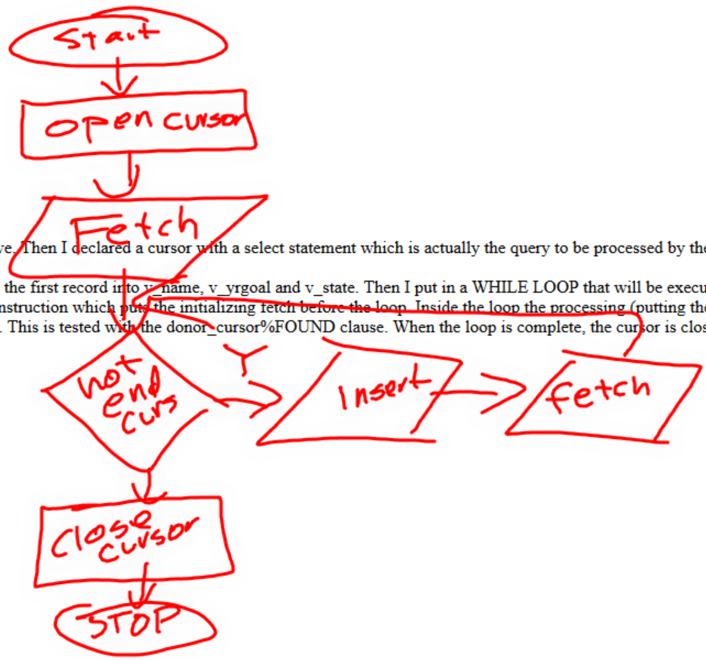
```
6 rows deleted.

SQL> select * from donor_part;

no rows selected
```

When using explicit cursor in PL/SQL there are four things that must be accomplished by the programmer.

- First, the cursor must be declared.
- Second, the cursor needs to be opened.
- Third, the fetch is used to fetch the results of the query into the variables declared in PL/SQL.
- Fourth, the cursor needs to be closed.

In this PL/SQL block I declared three variables v_name, v_yrgoal and v_state to correspond to the three fields in the new table that I created above. Then I declared a cursor with a select statement which is actually the query to be processed by the cursor.

In the begin, I open the declared cursor and then fetch the first record into the variables that I declared. This will put name, yrgoal, and state from the first record into v_name, v_yrgoal and v_state. Then I put in a WHILE LOOP that will be executed as long as the donor_cursor finds a record. Inside the loop I insert the data into the record into the new table and then fetch the next record. Notice the construction which puts the initializing fetch before the loop. Inside the loop the processing (putting the record into the new file) is done and then the next fetch is executed. The WHILE LOOP is done as long as the fetch is successful and a record is fetched. This is tested with the donor_cursor%FOUND clause. When the loop is complete, the cursor is closed. Looking at the results you can see that there are now six records in the new table that correspond to the six records in the donor table.



SQL CODE:

```
SQL> edit cursor1
```

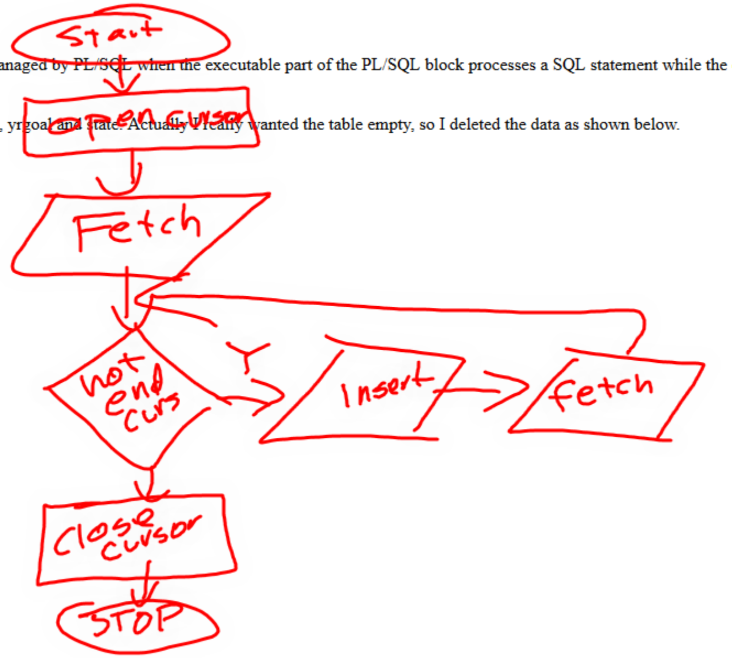
PL/SQL CODE:

```
DECLARE
  v_name     donor.name%TYPE;
  v_yrgoal   donor.yrgoal%TYPE;
  v_state    donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
WHILE donor_cursor%FOUND LOOP
  INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
END LOOP;
CLOSE donor_cursor;
END;
/
```

SQL CODE:

```
SQL> @ cursor1
Input truncated to 1 characters

PL/SQL procedure successfully completed.

SQL> select * from donor_part;
```

Cursors in PL/SQL



Oracle supports two types of cursors that can be used in PL/SQL: the implicit cursor and the explicit cursor. The implicit cursor is created and managed by PL/SQL when the executable part of the PL/SQL block processes a SQL statement while the explicit cursor is created and managed by the programmer.

Before I demonstate the use of the cursor, I wanted a table that contained parts of the donor table. I created the donor_part table to hold the name, yrgoal and state. Actually I really wanted the table empty, so I deleted the data as shown below.

SQL CODE:

```
SQL> CREATE TABLE donor_part
  2  (name, yrgoal, state)
  3  AS SELECT name, yrgoal, state FROM donor;

Table created.

SQL> SELECT * FROM donor_part;

NAME              YRGOAL ST
--------------- --------- --
Stephen Daniels       500 MA
Jennifer Ames         400 RI
Carl Hersey               RI
Susan Ash             100 MA
Nancy Taylor           50 MA
Robert Brooks          50 MA

6 rows selected.

SQL> delete from donor_part;

6 rows deleted.

SQL> select * from donor_part;

no rows selected
```

When using explicit cursor in PL/SQL there are four things that must be accomplished by the programmer.

- First, the cursor must be declared.
- Second, the cursor needs to be opened.
- Third, the fetch is used to fetch the results of the query into the variables declared in PL/SQL.
- Fourth, the cursor needs to be closed.

In this PL/SQL block I declared three variables v_name, v_yrgoal and v_state to correspond to the three fields in the new table that I created above. Then I declared a cursor with a select statement which is actually the query to be processed by the cursor.

In the begin, I open the declared cursor and then fetch the first record into the variables that I declared. This will put name, yrgoal, and state from the first record into v_name, v_yrgoal and v_state. Then I put in a WHILE LOOP that will be executed as long as the donor_cursor finds a record. Inside the loop I insert the data into the record into the new table and then fetch the next record. Notice the construction which puts the initializing fetch before the loop. Inside the loop the processing (putting the record into the new file) is done and then the next fetch is executed. The WHILE LOOP is done as long as the fetch is successful and a record is fetched. This is tested with the donor_cursor%FOUND clause. When the loop is complete, the cursor is closed. Looking at the results you can see that there are now six records in the new table that correspond to the six records in the donor table.

Type here to search
9:59 AM
10/29/2019

File Edit View History Bookmarks Tools Help

Introduction to cursors

www.pgrocer.net/Cis50/cursors.html

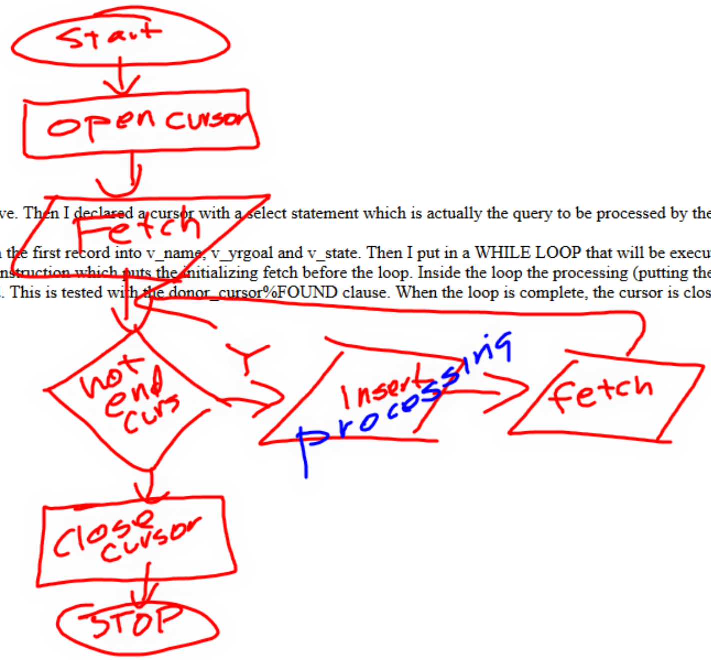
```
SQL> select * from donor_part;

no rows selected
```

When using explicit cursor in PL/SQL there are four things that must be accomplished by the programmer.

- First, the cursor must be declared.
- Second, the cursor needs to be opened.
- Third, the fetch is used to fetch the results of the query into the variables declared in PL/SQL.
- Fourth, the cursor needs to be closed.

In this PL/SQL block I declared three variables v_name, v_yrgoal and v_state to correspond to the three fields in the new table that I created above. Then I declared a cursor with a select statement which is actually the query to be processed by the cursor.

In the begin, I open the declared cursor and then fetch the first record into the variables that I declared. This will put name, yrgoal, and state from the first record into v_name, v_yrgoal and v_state. Then I put in a WHILE LOOP that will be executed as long as the donor_cursor finds a record. Inside the loop I insert the data into the record into the new table and then fetch the next record. Notice the construction which puts the initializing fetch before the loop. Inside the loop the processing (putting the record into the new file) is done and then the next fetch is executed. The WHILE LOOP is done as long as the fetch is successful and a record is fetched. This is tested with the donor_cursor%FOUND clause. When the loop is complete, the cursor is closed. Looking at the results you can see that there are now six records in the new table that correspond to the six records in the donor table.



SQL CODE:

```
SQL> edit cursor1
```

PL/SQL CODE:

```
DECLARE
  v_name     donor.name%TYPE;
  v_yrgoal   donor.yrgoal%TYPE;
  v_state    donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
WHILE donor_cursor%FOUND LOOP
  INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
END LOOP;
CLOSE donor_cursor;
END;
/
```

processing

SQL CODE:

```
SQL> @ cursor1
Input truncated to 1 characters

PL/SQL procedure successfully completed.

SQL> select * from donor_part;

NAME                YRGOAL ST
```

Type here to search

10:02 AM 10/29/2019

Introduction to cursors

www.pgrocer.net/Cis50/cursors.html

PL/SQL CODE:

```
DECLARE
  v_name     donor.name%TYPE;
  v_yrgoal   donor.yrgoal%TYPE;
  v_state    donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
WHILE donor_cursor%FOUND LOOP
  IF v_yrgoal > 50 THEN
    INSERT INTO donor_part
      VALUES(v_name, v_yrgoal, v_state);
  END IF;
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
END LOOP;
CLOSE donor_cursor;
END;
/
```

I am only inserting certain records.
See how this can be handled differently in the cursor on the next slide.

SQL CODE:

```
SQL> @ cursor2
Input truncated to 1 characters

PL/SQL procedure successfully completed.

SQL> SELECT * FROM donor_part;

NAME                YRGOAL ST
--------------- ---------- --
Stephen Daniels        500 MA
Jennifer Ames          400 RI
Susan Ash              100 MA
```

I can also use the SELECT in the cursor to accomplish the goal of only showing selected records. In this example I added WHERE yrgoal > 50 to the SELECT. Notice then when I test these PL/SQL blocks, I need to clean out donor_part prior to the test or I will see the results of the previous test in addition to the current test.

SQL CODE:

```
SQL> delete from donor_part;

6 rows deleted.

SQL> edit cursor2a
```

Title: Oct 29-10:04 AM (13 of 18)

File Edit View History Bookmarks Tools Help

Introduction to cursors

www.pgrocer.net/Cis50/cursors.html

test these PL/SQL blocks, I need to clean out donor_part prior to the test or I will see the results of the previous test in addition to the current test.

SQL CODE:

```
SQL> delete from donor_part;

6 rows deleted.

SQL> edit cursor2a
```

PL/SQL CODE:

```
DECLARE
  v_name     donor.name%TYPE;
  v_yrgoal   donor.yrgoal%TYPE;
  v_state    donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor
      WHERE yrgoal> 50;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
WHILE donor_cursor%FOUND LOOP
  INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
END LOOP;
CLOSE donor_cursor;
END;
/
```

Now the cursor will only contain records where yrgoal is greater than 50.

SQL CODE:

```
SQL> @ cursor2a
Input truncated to 1 characters

PL/SQL procedure successfully completed.

SQL> SELECT * FROM donor_part;

NAME               YRGOAL ST
--------------- --------- --
Stephen Daniels       500 MA
Jennifer Ames         400 RI
Susan Ash             100 MA
```

Type here to search

10:09 AM
10/29/2019

Title: Oct 29-10:09 AM (14 of 18)

Introduction to cursors

www.pgrocer.net/Cis50/cursors.html

```
SQL> delete from donor_part;

6 rows deleted.

SQL> edit cursor2a
```

name of script

PL/SQL CODE:

```
DECLARE
  v_name     donor.name%TYPE;
  v_yrgoal   donor.yrgoal%TYPE;
  v_state    donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor
      WHERE yrgoal> 50;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
WHILE donor_cursor%FOUND LOOP
  INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
END LOOP;
CLOSE donor_cursor;
END;
/
```

Script

SQL CODE:

executing script

```
SQL> @ cursor2a
Input truncated to 1 characters

PL/SQL procedure successfully completed.

SQL> SELECT * FROM donor_part;

NAME               YRGOAL ST
--------------- --------- --
Stephen Daniels       500 MA
Jennifer Ames         400 RI
Susan Ash             100 MA
```

We will be looking at four SQL cursor attributes that can be very useful in programming because they allow for verification of the results of DML statements. The SQL%ROWCOUNT tells how many rows were effected by the last SQL statement that was executed and can be used to only process a certain number of rows. The SQL%FOUND which was illustrated above, gives a boolean TRUE if the last SQL statement effected rows and the SQL%NOTFOUND gives a boolean true if the last SQL statement did not affect any rows. The %FOUND and %NOTFOUND

Type here to search

10:19 AM
10/29/2019

File Edit View History Bookmarks Tools Help
Introduction to cursors
www.pgrocer.net/Cis50/cursors.html

boolean true if the last SQL statement did not affect any rows. The %FOUND and %NOTFOUND are very useful for testing when a loop is complete. The SQL%ISOPEN will only fetch rows if the cursor is open. For example, you could do the following test in your code: IF NOT donor_cursor%ISOPEN THEN. Inside the loop you could open the cursor and do any other processing needed.

In the examples above I have been using the WHILE loop (my primary choice). You can also use the other loop choices discussed in the loop section. In this example I will do the program above using the EXIT WHEN and test using the %NOTFOUND. The syntax is LOOP which includes the processing. The last command inside the loop will say EXIT WHEN the name of the cursor%NOTFOUND meaning exit the loop when the fetch does not yield a row to be processed.

SQL CODE:

```
SQL> delete from donor_part;

4 rows deleted.

SQL> edit cursor2b
```

Note the different loop and the exit when the cursor is not found.

PL/SQL CODE:

```
DECLARE
  v_name     donor.name%TYPE;
  v_yrgoal   donor.yrgoal%TYPE;
  v_state    donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor
      WHERE yrgoal> 50;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
LOOP
  INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
  EXIT WHEN donor_cursor%NOTFOUND;
END LOOP;
CLOSE donor_cursor;
END;
/
```

SQL CODE:

```
SQL> @ cursor2b
Input truncated to 1 characters
```

Type here to search
10:32 AM
10/29/2019

File Edit View History Bookmarks Tools Help

Introduction to cursors

www.pgrocer.net/Cis50/cursors.html

```
PL/SQL procedure successfully completed.

SQL> SELECT * FROM donor_part;

NAME               YRGOAL ST
--------------- --------- --
Stephen Daniels       500 MA
Jennifer Ames         400 RI
Susan Ash             100 MA
```

In the next example, I only want to process the first 4 rows in the donor table. I can handle this using ROWCOUNT. As always I should always cover all my bases by testing for FOUND as well in case there are less than 4 rows in the table. Note that these two conditions are in an AND relationship because I only want to continue processing if both of them are true.

SQL CODE:

```
SQL> edit cursor3
```

PL/SQL CODE:

```
DECLARE
  v_name     donor.name%TYPE;
  v_yrgoal   donor.yrgoal%TYPE;
  v_state    donor.state%TYPE;
  CURSOR donor_cursor IS
    SELECT name, yrgoal, state
      FROM donor;
BEGIN
OPEN donor_cursor;
FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
WHILE donor_cursor%ROWCOUNT < 5 AND donor_cursor%FOUND LOOP
  INSERT INTO donor_part
    VALUES(v_name, v_yrgoal, v_state);
  FETCH donor_cursor INTO v_name, v_yrgoal, v_state;
END LOOP;
CLOSE donor_cursor;
END;
/
```

Exit the loop based on rowcount and if there are not enough records to cause this exit, exit when the cursor is not found.

SQL CODE:

```
SQL> @ cursor3

PL/SQL procedure successfully completed.
```

Type here to search

10:34 AM
10/29/2019

Title: Oct 29-10:34 AM (17 of 18)

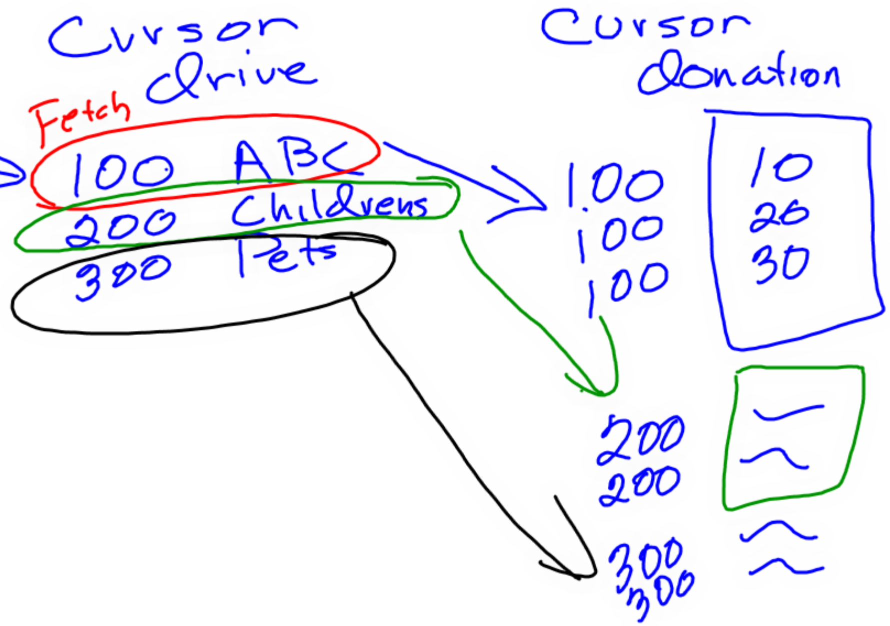
File Edit View History Bookmarks Tools Help

First cursor example

www.pgrocer.net/Cis50/cursorexampl1.html

PL/SQL CODE:

```
SET SERVEROUTPUT ON
DECLARE
  v_drive_no      drive.driveno%TYPE;
  v_drive_name    drive.drivename%TYPE;
  v_contamt       donation.contamt%TYPE;
  v_tot_contamt   cont_info.contamt%TYPE;
  CURSOR drive_cursor IS
     SELECT driveno, drivename FROM drive
     ORDER BY driveno;
  CURSOR donation_cursor IS
     SELECT contamt FROM donation
     WHERE v_drive_no = driveno
     ORDER BY driveno;
  BEGIN
     OPEN drive_cursor;
     LOOP
       FETCH drive_cursor INTO v_drive_no, v_drive_name;
       EXIT WHEN drive_cursor%NOTFOUND;
       IF donation_cursor%ISOPEN THEN
         CLOSE donation_cursor;
       END IF;
       OPEN donation_cursor;
       v_tot_contamt := 0;
       LOOP
         FETCH donation_cursor INTO v_contamt;
         EXIT WHEN donation_cursor%NOTFOUND;
         v_tot_contamt := v_tot_contamt + v_contamt;
         dbms_output.put_line('The current amount is: '||v_tot_contamt);
       END LOOP;
       INSERT into cont_info
          VALUES(v_drive_no, v_drive_name, v_tot_contamt);
       CLOSE donation_cursor;
     END LOOP;
     CLOSE drive_cursor;
END;
/
SET SERVEROUTPUT OFF
```



SQL CODE:

```
SQL> @ cursor6a
The current amount is: 25
The current amount is: 45
The current amount is: 55
The current amount is: 105
The current amount is: 40
The current amount is: 75
The current amount is: 10
The current amount is: 20

PL/SQL procedure successfully completed.

Input truncated to 20 characters
SQL> SELECT * FROM cont_info;

DRI DRIVENAME          CONTAMT
```

Type here to search

10:49 AM
10/29/2019

Title: Oct 29-10:49 AM (18 of 18)